Scalable Informatics

Informatics on AMD64

An investigation of observed performance on informatics and chemistry programs

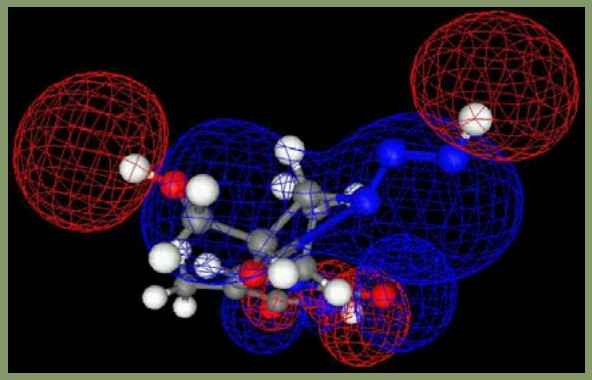
Joseph Landman, Ph.D.
Founder and CEO, Scalable Informatics LLC
landman@scalableinformatics.com
http://scalableinformatics.com/

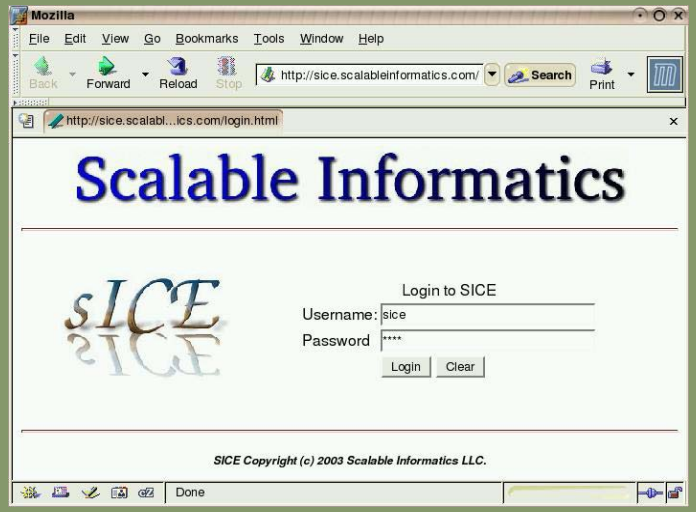
Agenda

- Introduction
- Observations and analysis
 - Overview
 - BLAST
 - HMMer
 - Chemistry programs
- Conclusion
- Questions

Introduction

- Scalable Informatics
 - Focus upon
 - **High performance** informatics software
 - Cluster **design & integration**
 - Cluster tools
 - Code porting, optimization, & parallelization
 - Product development & marketing
 - Collaboration





Agenda

- Introduction
- Observations and analysis
 - Overview
 - BLAST
 - HMMer
 - Chemistry programs
- Conclusion
- Questions

Research computing users observed ...

- **Performance benefit to recompiling for 64-bit mode**
 - For specific codes
 - Not well-documented (more anecdotal)
 - Not well-characterized
- **What was going on?**
 - ***Where is it faster*** than 32 bit mode?
 - ***Why*** is it faster?
 - ***What applications*** is this effect observed in?
 - ***Need to perform tests under controlled conditions ...***

Specifically

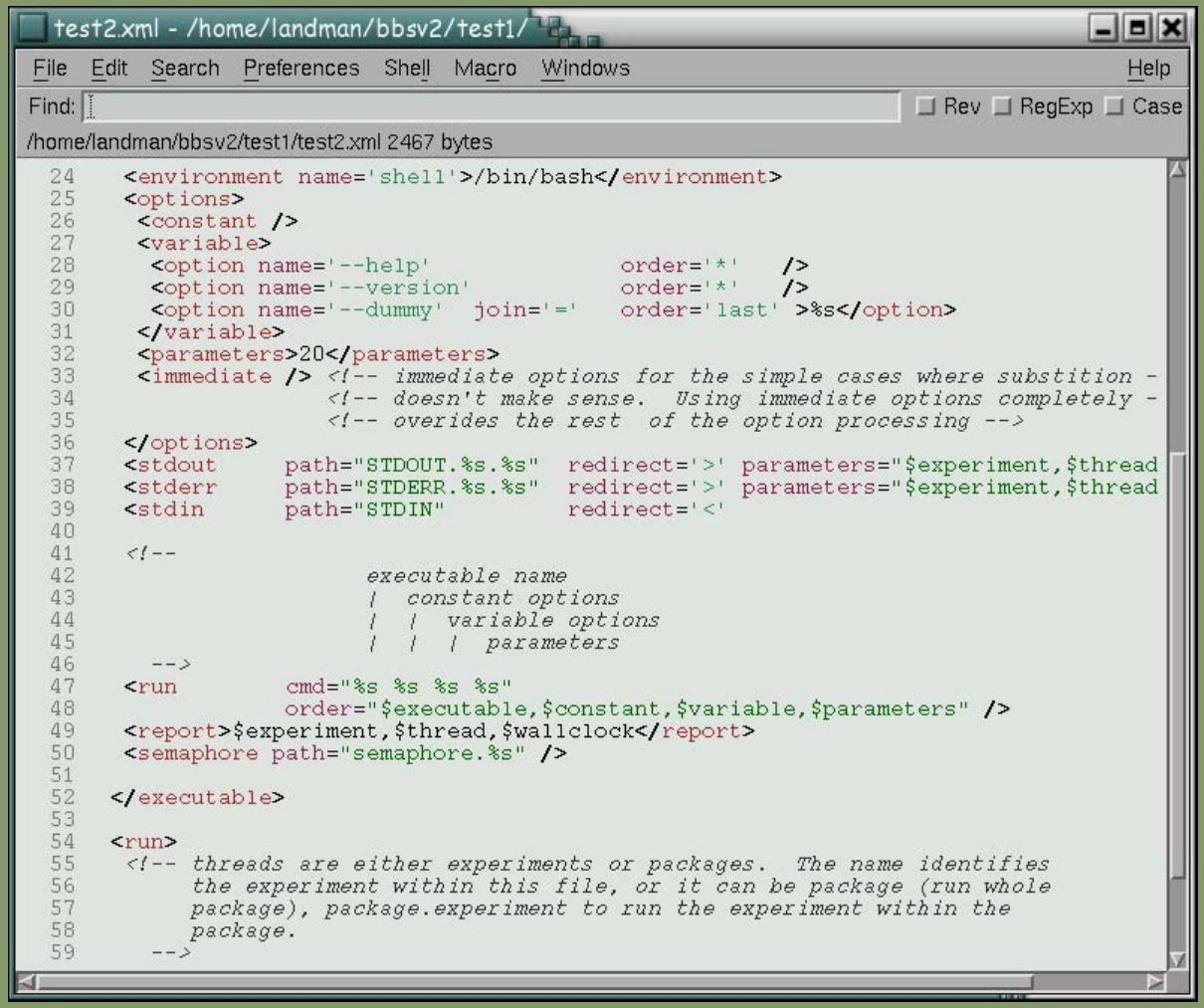
- **Benchmarks using BBS**
 (http://bioinformatics.org/bbs/)
 - Blastn slight disadvantage
 - Blastx slight advantage
 - tblast{x,n} significant advantage





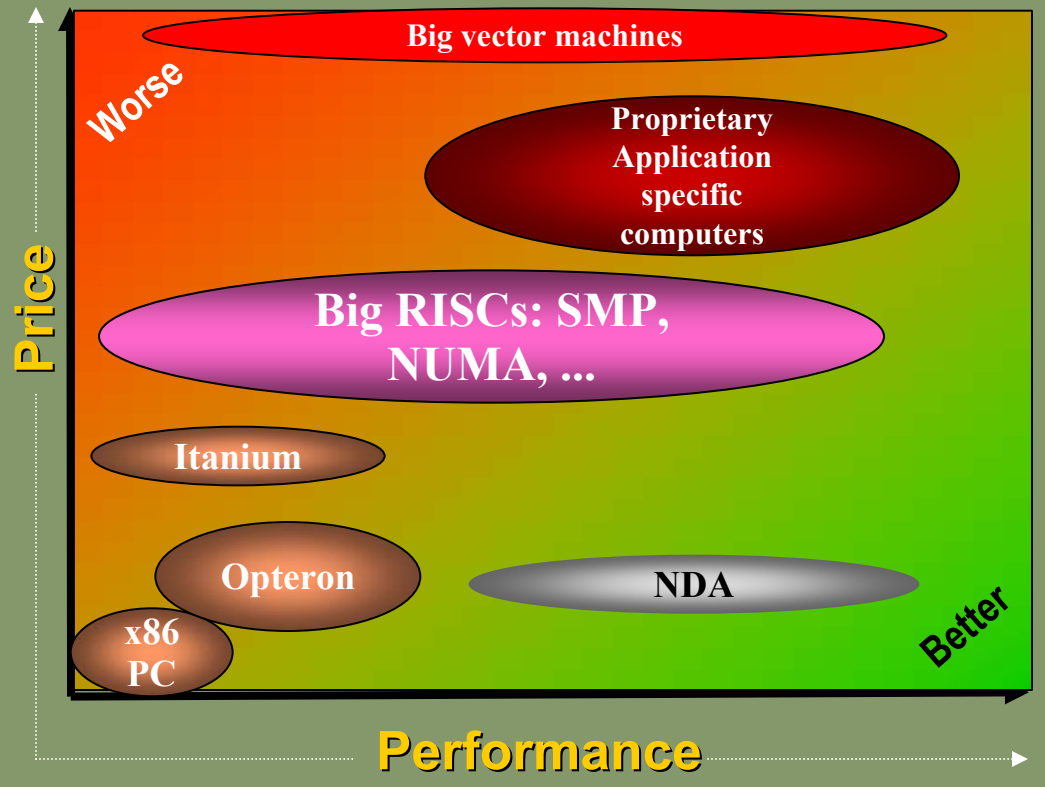
BBS – A benchmark framework

- **Goals:**
 - Meaningful results
 - Repeatable results
 - simple/complex tests
- **Features:**
 - Platform independent
 - Binary independent
- **v2.x coming soon ...**

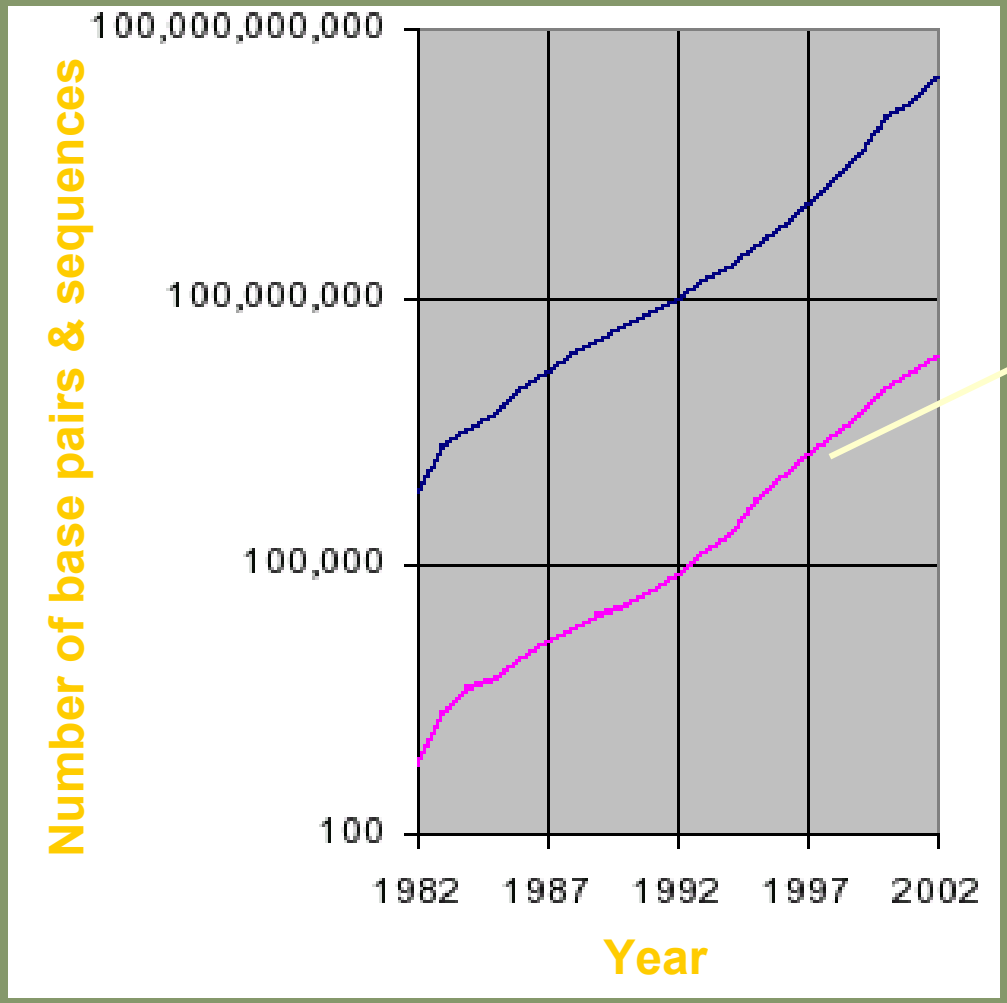


Why is this interesting?

- Informatics users need
 - ***More*** (integer) ***performance***
 - Large memory
 - Large I/O capability
- Chemistry users need
 - ***More*** (floating point) ***performance***
 - Large memory
 - Large I/O capability

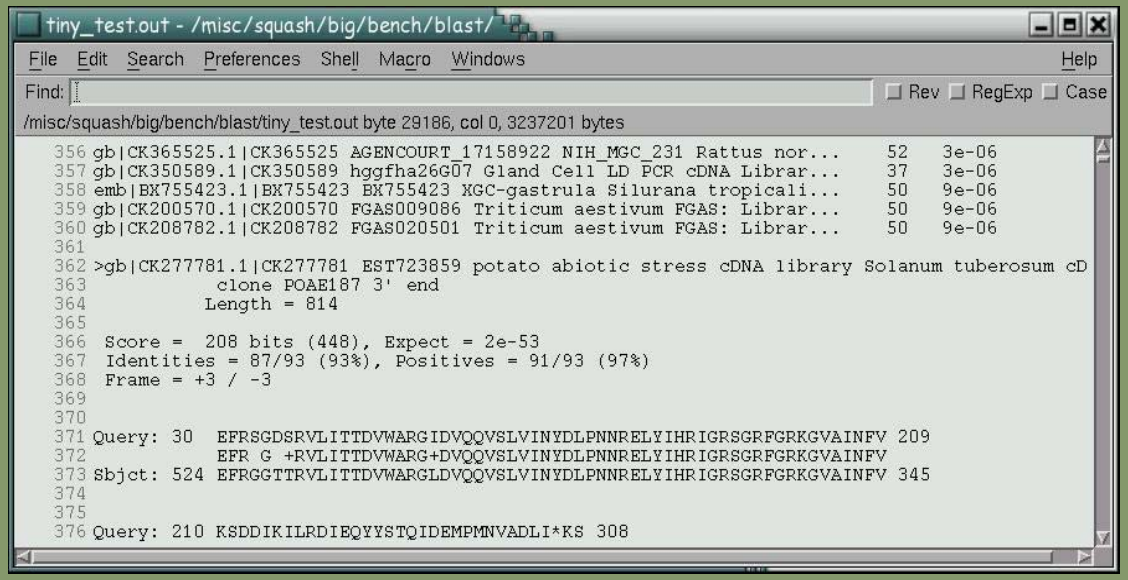


Data growth rates



Initial observations

- **Existing IA32 binaries run**
 - Occasional surprises in compiling
 - Usually architecture detection
 - Performance inline with expectations
- **Compiling for 64 bit**
 - Fairly easy (-m64)
 - Most codes work



The task ...

- Create a controlled enviroment
- Build test cases
- Measure performance
- Explain

Platforms for tests

- **OS**
 - **SuSE 9.0**
 - mature for AMD64
 - **Fedora core** 1 for AMD64
 - Others
 - Gentoo 2004.1
 - Tao/cAos
- **Compilers**
 - GCC
 - 3.3.1
 - 3.3.3
 - 3.4-cvs
- **Hardware**
 - Tyan s2880
 - 2x **Opteron** 246
 - 2 GB ram (DDR 333)
 - 2x SATA RAID 0 stripe

Test cases

- **Used BBS for NCBI BLAST**
 - Modified to include multiple algorithm tests
- **Used included test cases for**
 - HMMer (globin50 vs nr)
 - Moldy (water, clay)
 - Md43 (NaCl)

NCBI BLAST observations

- **Had to patch**
 - **2.2.6** and **2.2.7** source
 - To recognize architecture
 - **2.2.8** linux-x86_64.ncbi.mk
 - To recognize architecture
 - To set compiler options correctly
- **Built RPMs**
 - Compiled m32 and m64, as well as register pressured*

*** used -ffixed-r12 -ffixed-r13 -ffixed-r14 -ffixed-r15 to restrict available registers for instruction schedule to use**

All patches and RPMs available from
http://downloads.scalableinformatics.com/downloads/ncbi/opteron/

Profile …

- **Sharp peaks**
 - 95% of time spent in small group of routines
 - Independent of ABI
 - 85% of time in Blastwordfinder_mh_contig
 - Varied a few % depending upon the algorithm used
 - Independent of ABI

~10% of time spent in this loop:

```
do {
     /* lookup a contiguous word. */
     s++;
     lookup_index = next_lindex;
     if (s == s_end) goto NormalReturn;
     next_lindex = (((lookup_index) &
          mask)<<char_size) + *(s+1);
     next_pv_array_addr = &pv_array
          [next_lindex>>PV_ARRAY_BTS];
     pv_val = next_pv_val;
     next_pv_val = *next_pv_array_addr;
    } while ((pv_val&(((PV_ARRAY_TYPE) 1)
           << (lookup_index &
               PV_ARRAY_MASK))) == 0);
num_hits = mod_lt[lookup_index].num_used;
```

~20% of time spent in similar loops

Profile

32-bit

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ks/call  Ks/call  name
 76.82   1226.53  1226.53 285541639     0.00   0.00  BlastExtendWordSearch
 11.41   1408.64   182.11 285541638     0.00   0.00  BlastTranslateUnambiguousSequence
  4.30   1477.34    68.70 47590272     0.00   0.00  BLASTPerformFinalSearch
  4.06   1542.16    64.82 310887726    0.00   0.00  BlastWordExtend_prelim
```

64-bit

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ks/call  Ks/call  name
 80.80   1073.64  1073.64 285541603    0.00     0.00 BlastExtendWordSearch
  8.11   1181.45   107.81 285541623    0.00     0.00 BlastTranslateUnambiguousSequence
  4.21   1237.43    55.98 310887725    0.00     0.00  BlastWordExtend_prelim
  3.38   1282.36    44.93 47590274     0.00     0.00  BLASTPerformFinalSearch
```

Measurements

test	Number of CPUs	32 bit time	64 bit time	64 bit advantage (deficit)
blastx	1	1,516	1,085	**28.4**
blastx	2	751	550	**26.8**
blastn	1	297	252	**15.2**
blastn	2	153	132	**13.7**
tblastx	1	4,999	3,545	**29.1**
tblastx	2	2,761	1,940	**29.7**

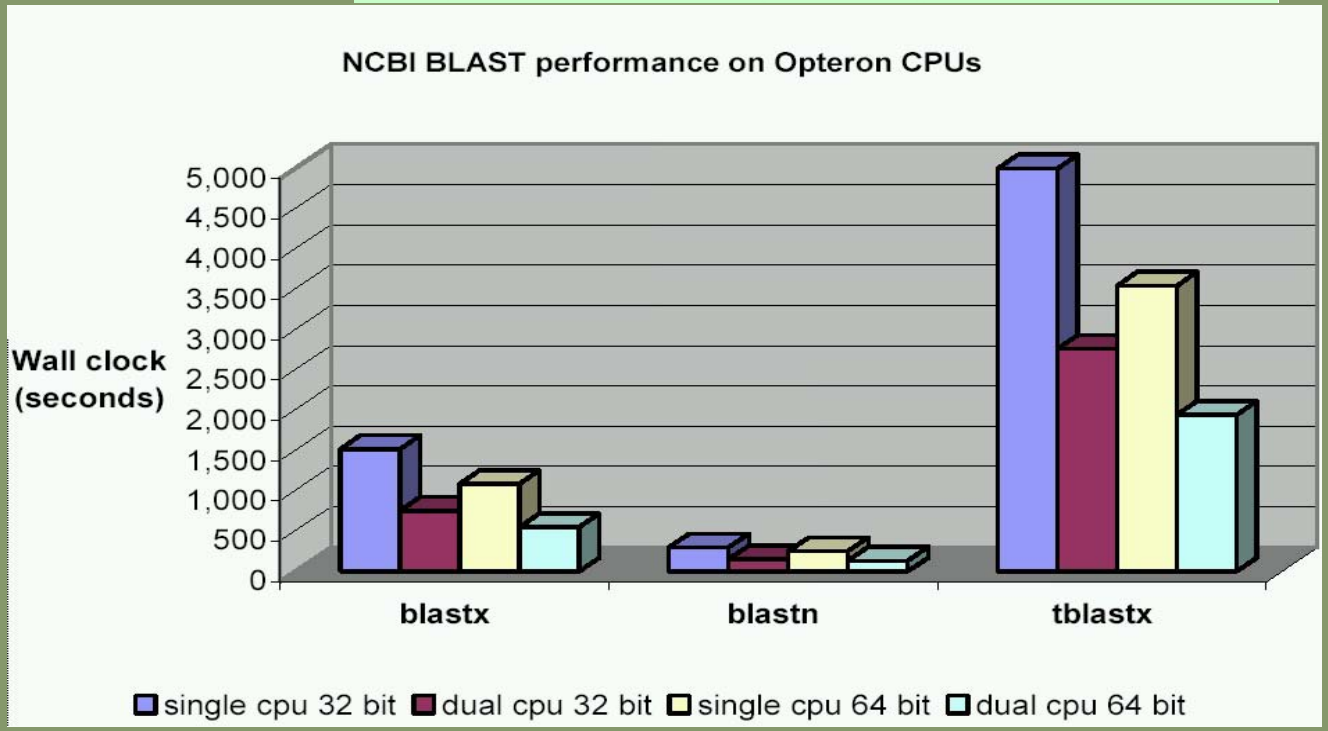
NCBI BLAST 2.2.8

- 74 tomato EST's

Blastn -> patnt

Blastx -> nr

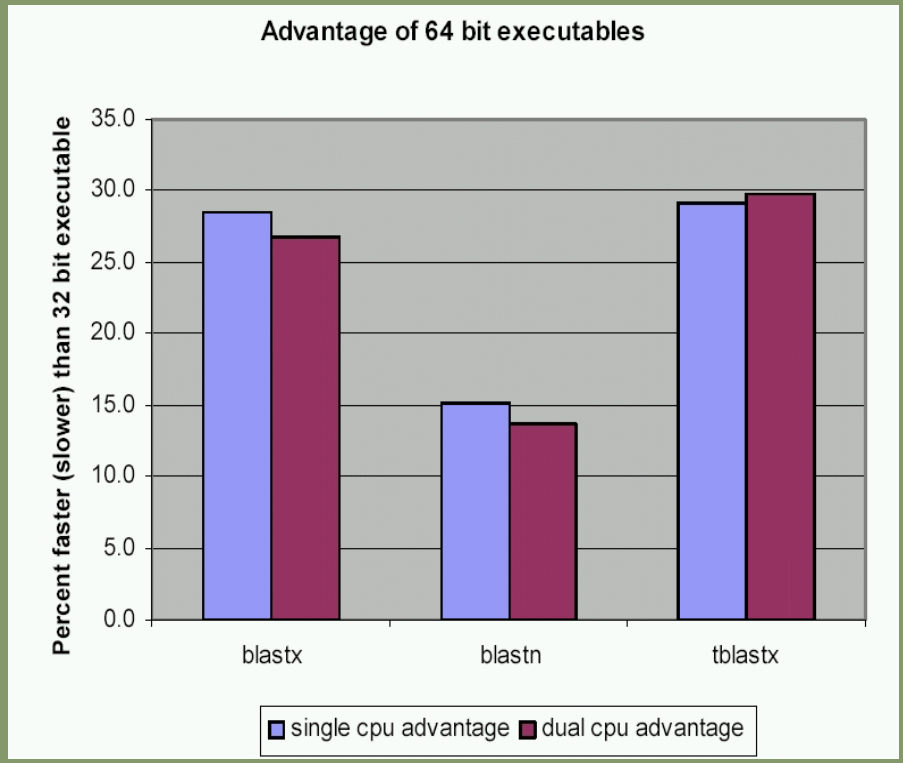
Tblastx-> patnt



Measurements

- Same data with register pressure 64 bit binaries

test	NCPU	32 bit	64 bit	64 bit with register pressure
blastx	1	1,516	1,085	1,132
blastx	2	751	550	554
blastn	1	297	252	269
blastn	2	153	132	132



What is happening here?

- **Register pressure**
 - 64-bit has (compared to 32-bit)
 - 2x GPR
 - 2x SSE register
- **Compiler quality**
 - Same compiler
 - Different target ABI
- **Address space**
 - 64-bit -> flat
 - 32-bit -> segmented
- **Microarchitecture**
- **ABI (size of primative types)**
 - Should not be an issue
- **Instruction set**
 - 64-bit a superset
- **OS**
 - Observations consistent across
 - Distributions
 - Kernel versions
 - 2.4.19
 - 2.4.21
 - 2.6.5-rc2

- **Register pressure**
 - Increasing register
 - **Small performance decrease**
- **Microarchitecture**
 - Doesnt change
 - Different latencies in 32-bit vs 64-bit mode?
- **ABI**
 - **Would favor 32-bit** for fixed size cache
- **OS**
 - NUMA/affinity scheduling
 - Small performance changes
- **Compiler**
 - Gcc versions show little noticeable effect
 - Would like to test PGI, Intel, and PathScale

- **Address space**
 - 32-bit segmented
 - One extra operation per memory access
 - Larger op-code for indirect addressing
 - 64-bit flat
 - Simpler memory model
 - Smaller op-code for indirect addressing

Code inner loops do the same amount of work the same number of times across the ABIs

The expensive loop ...

```
Line     Code
7104 do {
7105     /* lookup a contiguous word. */
7106     s++;
7107     lookup_index = next_lindex;
7108
7109     if (s == s_end)
7110        goto NormalReturn;
7111
7112     next_lindex = (((lookup_index) & mask)<<char_size) + *(s+1);
7113     next_pv_array_addr = &pv_array[next_lindex>>PV_ARRAY_BTS];
7114     pv_val = next_pv_val;
7115     next_pv_val = *next_pv_array_addr;
7116
7117    } while ((pv_val&(((PV_ARRAY_TYPE) 1)<<(lookup_index&PV_ARRAY_MASK))) == 0);
```

32-bit assembly for expensive loop

```
/work/bench/ncbi/build/blast.c:7106
 805a290:       ff 85 34 fe ff ff       incl   0xfffffe34(%ebp)
/work/bench/ncbi/build/blast.c:7109
 805a296:       8b b5 9c fe ff ff       mov    0xfffffe9c(%ebp),%esi
 805a29c:       39 b5 34 fe ff ff       cmp    %esi,0xfffffe34(%ebp)
/work/bench/ncbi/build/blast.c:7107
 805a2a2:       8b 9d 60 fe ff ff       mov    0xfffffe60(%ebp),%ebx
/work/bench/ncbi/build/blast.c:7109
 805a2a8:       0f 84 ff 0a 00 00       je     805adad <BlastWordFinder_mh_contig+0xcdd>
/work/bench/ncbi/build/blast.c:7112
 805a2ae:       0f b6 8d 68 fe ff ff    movzbl 0xfffffe68(%ebp),%ecx
 805a2b5:       89 d8                   mov    %ebx,%eax
 805a2b7:       8b bd 34 fe ff ff       mov    0xfffffe34(%ebp),%edi
 805a2bd:       23 85 64 fe ff ff       and    0xfffffe64(%ebp),%eax
/work/bench/ncbi/build/blast.c:7115
 805a2c3:       8b 95 3c fe ff ff       mov    0xfffffe3c(%ebp),%edx
/work/bench/ncbi/build/blast.c:7112
 805a2c9:       d3 e0                   shl    %cl,%eax
 805a2cb:       0f b6 4f 01             movzbl 0x1(%edi),%ecx
/work/bench/ncbi/build/blast.c:7114
 805a2cf:       8b bd 38 fe ff ff       mov    0xfffffe38(%ebp),%edi
/work/bench/ncbi/build/blast.c:7112
 805a2d5:       01 c8                   add    %ecx,%eax
/work/bench/ncbi/build/blast.c:7117
 805a2d7:       89 d9                   mov    %ebx,%ecx
/work/bench/ncbi/build/blast.c:7112
 805a2d9:       89 85 60 fe ff ff       mov    %eax,0xfffffe60(%ebp)
/work/bench/ncbi/build/blast.c:7117
 805a2df:       83 e1 1f                and    $0x1f,%ecx
/work/bench/ncbi/build/blast.c:7113
 805a2e2:       c1 f8 05                sar    $0x5,%eax
/work/bench/ncbi/build/blast.c:7115
 805a2e5:       8b 34 82                mov    (%edx,%eax,4),%esi
```

64-bit assembly for the expensive loop

```
/work/bench/ncbi/build/blast.c:7106
  4128f0:       48 ff 85 c0 fc ff ff    incq   0xfffffffffffffcc0(%rbp)
/work/bench/ncbi/build/blast.c:7109
  4128f7:       4c 8b 8d 30 fd ff ff    mov    0xfffffffffffffd30(%rbp),%r9
  4128fe:       4c 39 8d c0 fc ff ff    cmp    %r9,0xfffffffffffffcc0(%rbp)
/work/bench/ncbi/build/blast.c:7107
  412905:       8b b5 04 fd ff ff       mov    0xfffffffffffffd04(%rbp),%esi
/work/bench/ncbi/build/blast.c:7109
  41290b:       0f 84 f7 09 00 00       je     413308
<BlastWordFinder_mh_contig+0xc38>
/work/bench/ncbi/build/blast.c:7112
  412911:       4c 8b b5 c0 fc ff ff    mov    0xfffffffffffffcc0(%rbp),%r14
  412918:       0f b6 8d 0c fd ff ff    movzbl 0xfffffffffffffd0c(%rbp),%ecx
  41291f:       89 f0                   mov    %esi,%eax
  412921:       23 85 08 fd ff ff       and    0xfffffffffffffd08(%rbp),%eax
/work/bench/ncbi/build/blast.c:7115
  412927:       4c 8b 9d e0 fc ff ff    mov    0xfffffffffffffce0(%rbp),%r11
/work/bench/ncbi/build/blast.c:7117
  41292e:       ba 01 00 00 00          mov    $0x1,%edx
/work/bench/ncbi/build/blast.c:7114
  412933:       4c 8b 95 d8 fc ff ff    mov    0xfffffffffffffcd8(%rbp),%r10
/work/bench/ncbi/build/blast.c:7112
```

```
/work/bench/ncbi/build/blast.c:7112
  41293a:       45 0f b6 6e 01          movzbl 0x1(%r14),%r13d
  41293f:       d3 e0                   shl    %cl,%eax
/work/bench/ncbi/build/blast.c:7117
  412941:       89 f1                   mov    %esi,%ecx
  412943:       83 e1 3f                and    $0x3f,%ecx
/work/bench/ncbi/build/blast.c:7112
  412946:       44 01 e8                add    %r13d,%eax
/work/bench/ncbi/build/blast.c:7117
  412949:       48 d3 e2                shl    %cl,%rdx
/work/bench/ncbi/build/blast.c:7112
  41294c:       89 85 04 fd ff ff       mov    %eax,0xfffffffffffffd04(%rbp)
/work/bench/ncbi/build/blast.c:7113
  412952:       c1 f8 06                sar    $0x6,%eax
/work/bench/ncbi/build/blast.c:7117
  412955:       4c 85 d2                test   %r10,%rdx
/work/bench/ncbi/build/blast.c:7113
  412958:       48 98                   cltq
/work/bench/ncbi/build/blast.c:7115
  41295a:       4d 8b 24 c3             mov    (%r11,%rax,8),%r12
  41295e:       4c 89 a5 d8 fc ff ff    mov    %r12,0xfffffffffffffcd8(%rbp)
/work/bench/ncbi/build/blast.c:7117
  412965:       74 89                   je     4128f0
<BlastWordFinder_mh_contig+0x220>
```

So why is NCBI BLAST faster on 64 bit?

- **Current thoughts:**
 - Better scheduling of 64-bit instructions
 - Not compiler register pressure, but ***more opportunity to rename registers for instructions in flight***
 - o-o-o and speculative execution
 - ***o-o-o helped by larger register file***
 - ***Simpler addressing*** mode

HMMer

- **Profile hidden Markov method**
 - Statistical pattern searches
 - Build a state machine
 - Apply to library of sequences/database
 - Return model fits above a threshold
 - Sequence alignment
- **Inexpensive**
 - Model building
- **Expensive**
 - Search
 - Pfamsearch
- **Code "vectorizes"**
 - Can be adapted to SIMD
 - Lots of register usage
 - Should be quite a bit of register pressure

Found a surprise

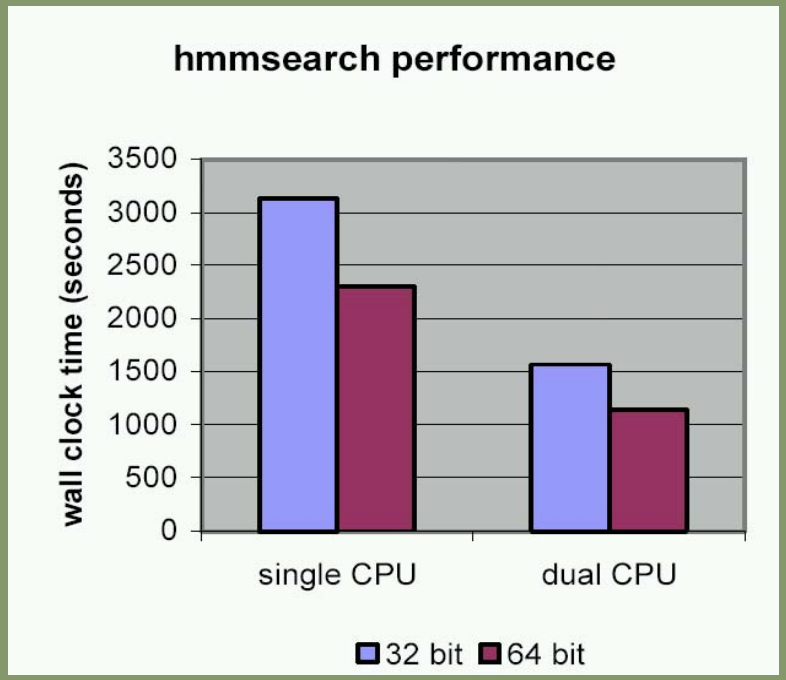
- Increasing register pressure increases performance

HMMer performance

- **Used globin.hmm example in HMMer documentation**
 - hmmsearch vs recent nr
 - 32 bit, 64 bit, and 64 bit increasing compiler register pressure
 - Fewer registers to build instruction schedule
 - Should be a detriment to the performance

-ffixed-r12 -ffixed-r13 \ -ffixed-r14 -ffixed-r15

NCPU	32 bit time	64 bit time	64 bit time with register pressure	Register pressure advantage	64 bit advantage (deficit) over 32 bit
1	3128.1	2935.31	2305.17	21.47	**26.3**
2	1566.47	1278.76	1139.36	10.9	**27.3**



Why does increasing pressure make HMMer go faster?

- **Too many allocated registers to effectively exploit**
 - Register renaming
 - o-o-o execution
 - Scheduler bubbles
- **Why is 32-bit mode slower?**
 - Smaller register file
 - Cannot use for o-o-o
 - Cannot use for register renaming
 - Fewer SSE2 registers

Chemistry

- Moldy
 - MM,MD, ...
- Md43
 - *ab-initio* MD
- Exhibit similar performance deltas for 32-bit vs 64-bit

Agenda

- Introduction
- Observations and analysis
 - Overview
 - BLAST
 - HMMer
 - Chemistry programs
- **Conclusion**
- Questions

Conclusions

- In the tested cases
 - ***64-bit mode is faster***
 - Register pressure matters
 - A little for BLAST
 - Much more for HMMer ***(it helps!)***
 - GCC not perfect
 - Model of cpu needs fine tuning
 - Better allocation decisions

Questions

- All copyrights, trademarks and brand names are property of their respective owners.